オペレーティングシステム

2022/10/18

海谷 治彦

目次

- プロセスについて

OSのプログラミングインタフェース

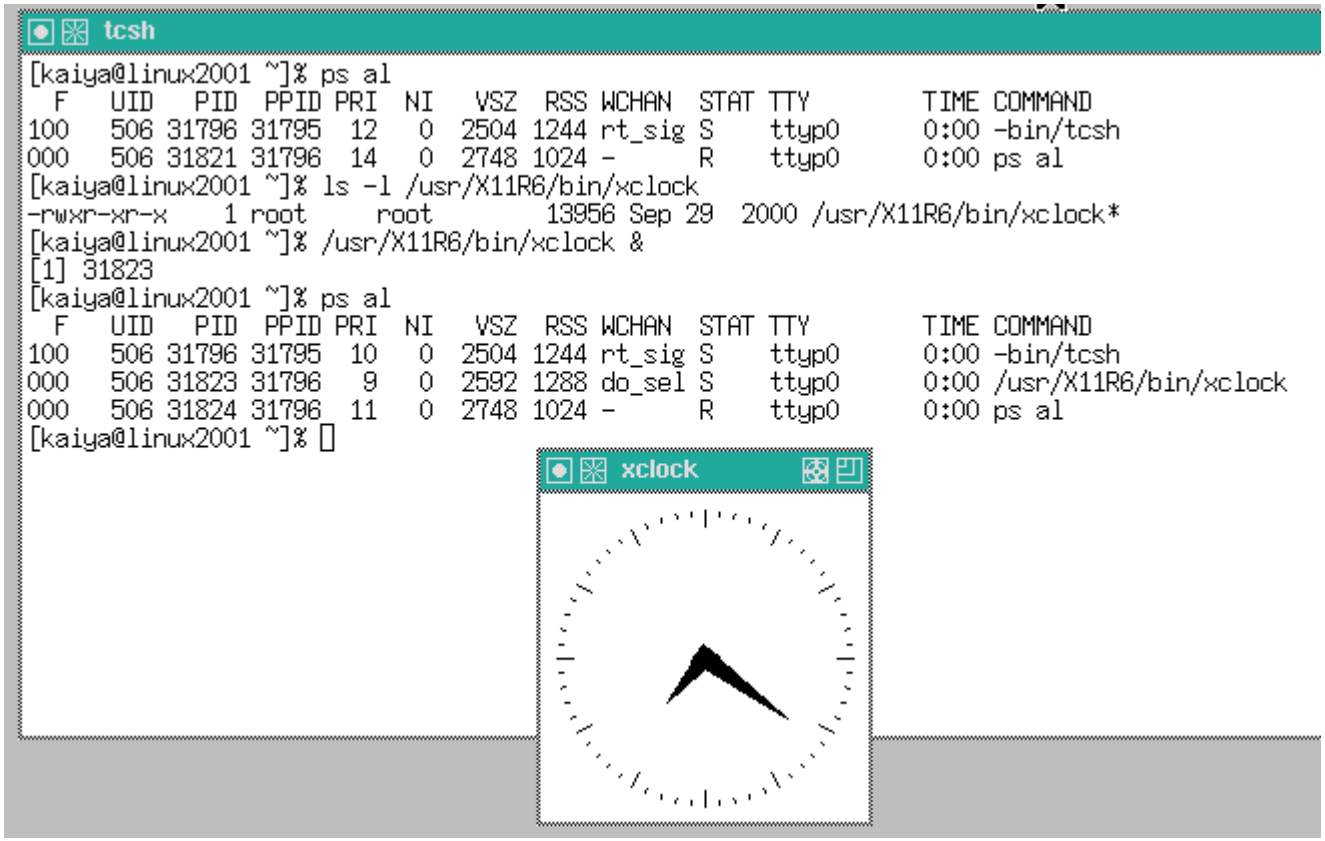
1. 目的
2. 提供
3. 具体的なAPI
4. 互換性と移植性

プロセスについて

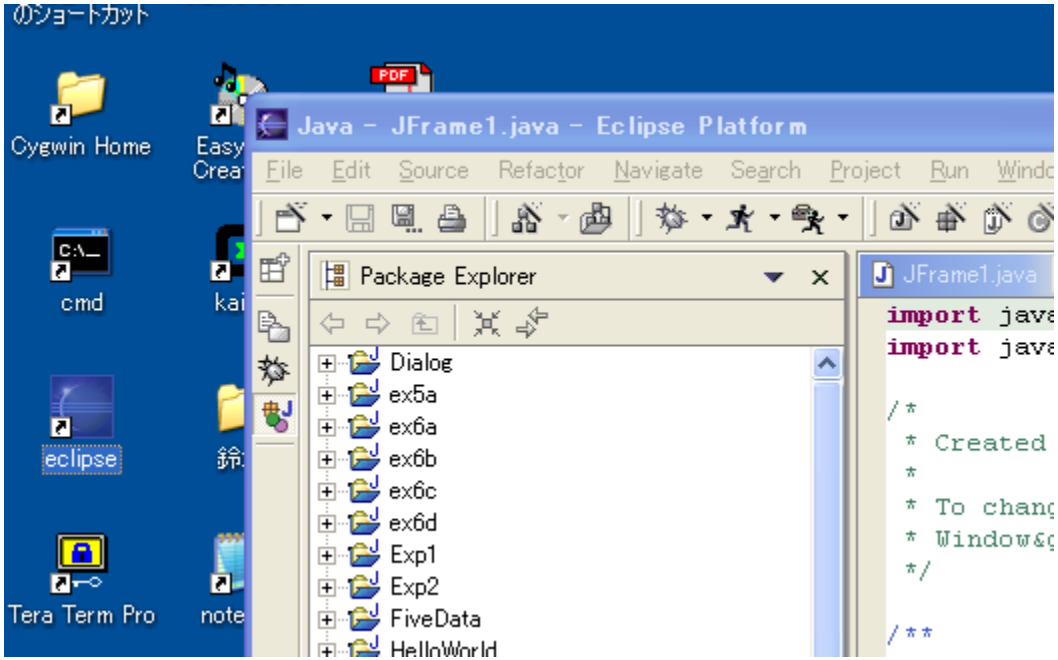
- 「プログラミ」ングインタフェースを語る前に，プログラムとは何でOSとの違うとは何について語る.
- 加えて，動作中のプログラムに相当するプロセスについて補足する.
- 先に結論
  - OSもアプリケーションもプログラムの一種
  - コンピュータの中には多数のアプリがプロセスとして動作し，(通常，)一個のOSが動作している.

アプリの動かし方 Linuxの例

ターミナルからコマンド名を打つ.

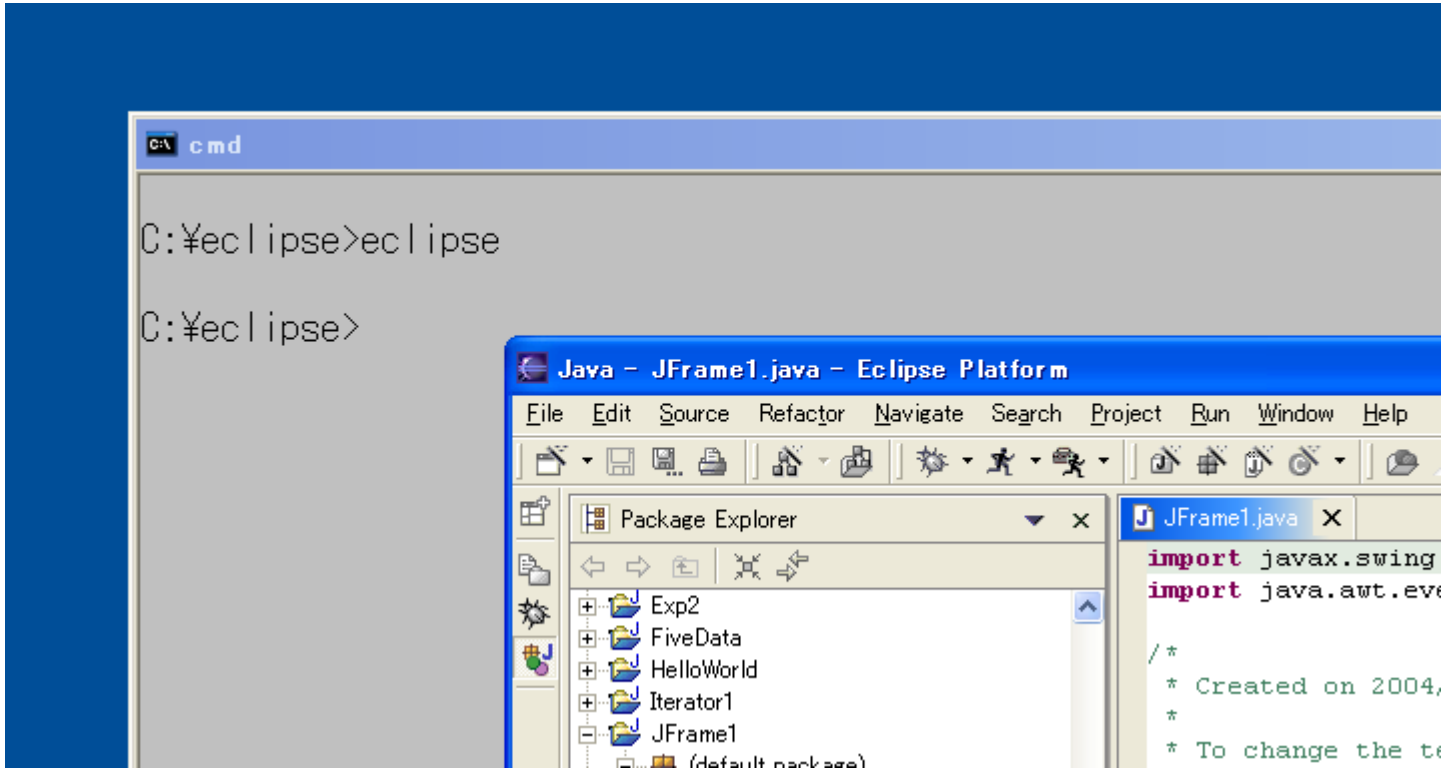


アプリの動かし方 Winの例1



基本的にはアイコンをつっつくとアプリが起動できる.

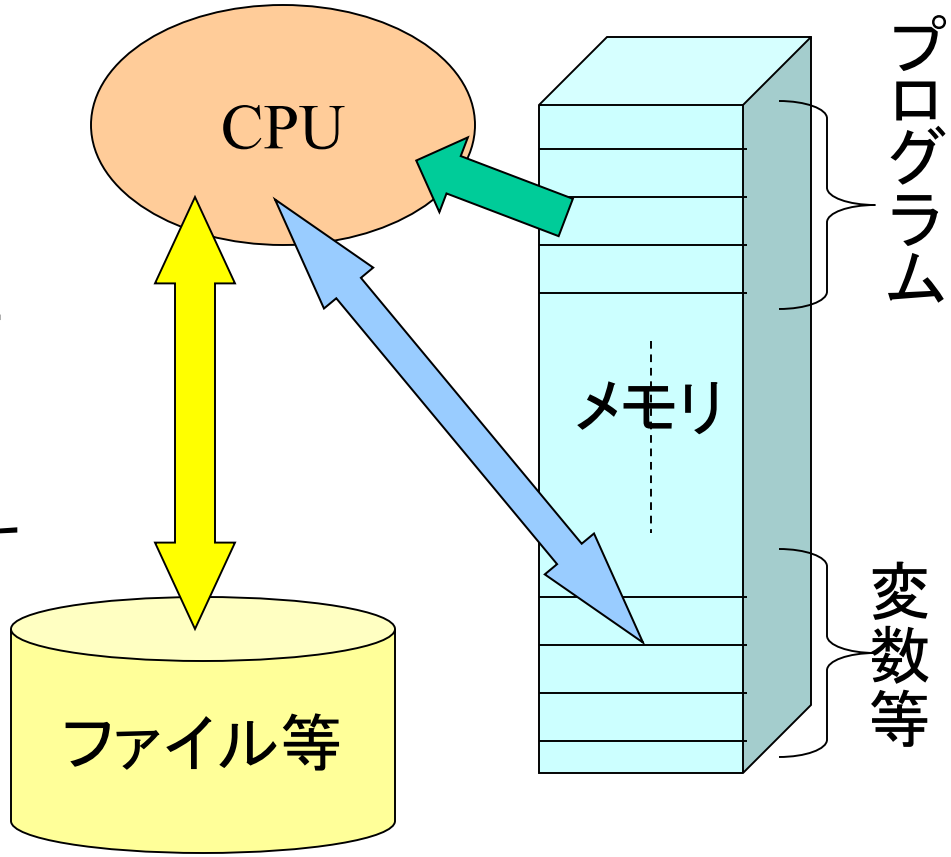
アプリの動かし方 Winの例2



実はWindowsでもコマンドを指定してプログラムを起動できる.

プログラムの処理の流れ

- プログラムがメモリに読み込まれる.
- 計算に必要なメモリも確保される.（変数等のため)
- CPUがプログラムを順に読んで，計算をする.
- 必要ならば，デバイス(ファイル等)にアクセスする.

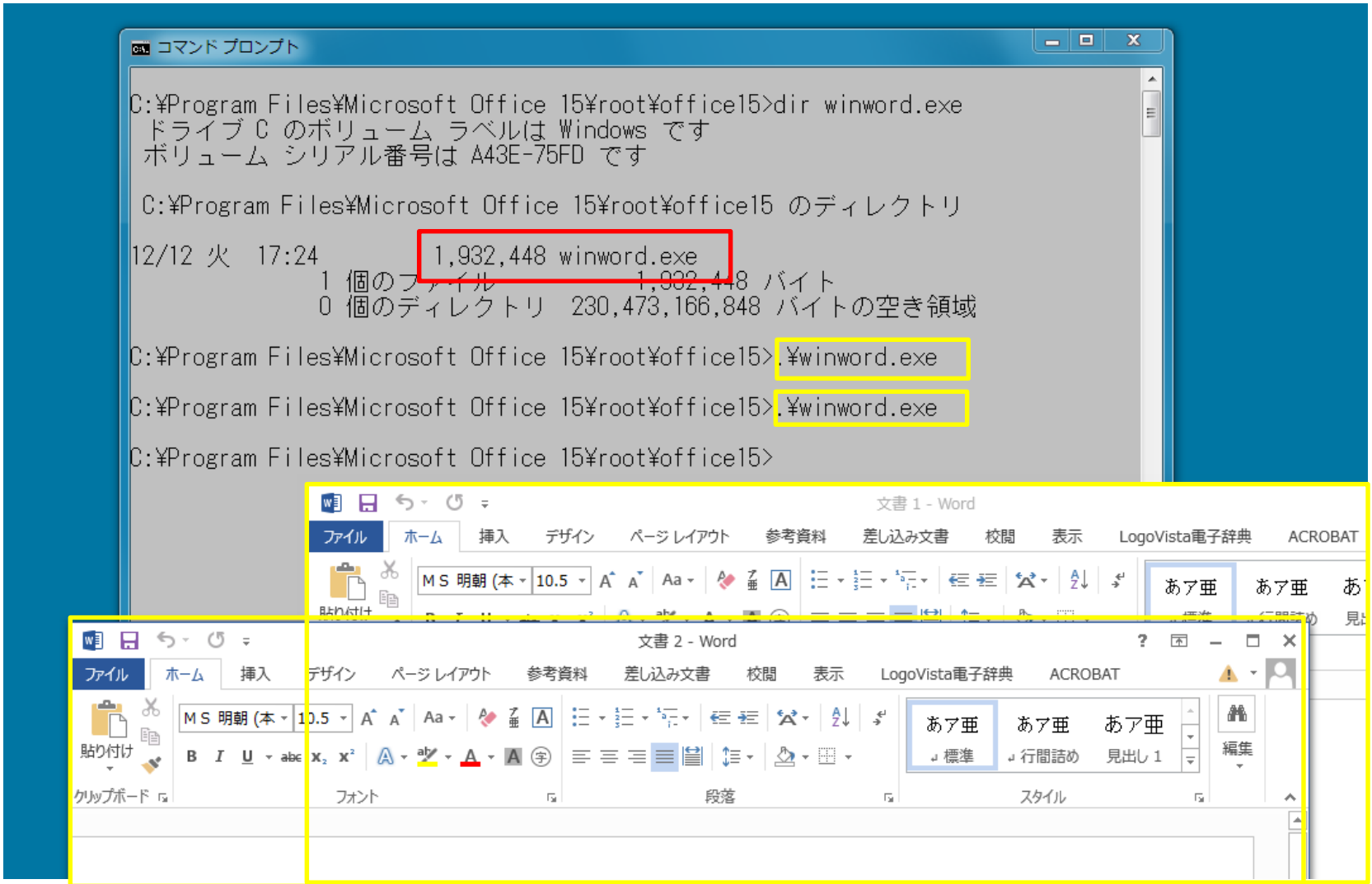


プロセス (タスク)

- 処理中のプログラム.
- プログラムのインスタンス.
- タスクとも呼ばれる.
- 前頁の「メモリに読み込まれたプログラムとデータ」に対応する概念.
- 1つのプログラムをもとに複数のプロセスが発生するため, プログラムとは概念的に区別される.
- 以下の対比でイメージを得て！

	書き物	実行
ソフトウェア	プログラム	プロセス
音楽	譜面	演奏
ゲーム	ルール(ブック)	実際のプレイ

プログラムとプロセス


コマンド プロンプト
C:¥Program Files¥Microsoft Office 15¥root¥office15>dir winword.exe
ドライブ C のボリューム ラベルは Windows です
ボリューム シリアル番号は A43E-75FD です
C:¥Program Files¥Microsoft Office 15¥root¥office15 のディレクトリ
12/12 火 17:24 1,932,448 winword.exe
1 個のファイル 1,932,448 バイト
0 個のディレクトリ 230,473,166,848 バイトの空き領域
C:¥Program Files¥Microsoft Office 15¥root¥office15>.¥winword.exe
C:¥Program Files¥Microsoft Office 15¥root¥office15>.¥winword.exe
C:¥Program Files¥Microsoft Office 15¥root¥office15>
文書 1 - Word
文書 2 - Word
ファイル
ホーム
挿入
デザイン
ページ レイアウト
参考資料
差し込み文書
校閲
表示
LogoVista電子辞典
ACROBAT
貼り付け
MS 明朝 (本
10.5
あア亜
標準
行間詰め
見出し 1
編集
クリップボード
フォント
段落
スタイル

プロセス動作について

- プロセスが計算を実行するためには資源が必要，少なくともメモリとCPUは必要.
- 1つのプログラムをもとに多数のプロセスが生成されている.
- 無論，実行可能なプログラムは多数ある.
- 沢山のプロセスが同時に動いている(ように見える).
- プロセスの生成と消滅が繰り返されている.

  OSを通して業務(アプリの実行)するので当たり前か.
- プロセスの寿命はまちまち.

  ls は一瞬で終わるが，httpd (ウエブサーバー)は何日も動いている．OSは当然，動きっぱなし.

プロセス管理のためのOSへの要件

- どんなプロセスが存在するのかを記録しておかなければならない.

  資源分配のため.
- プロセスの生成と削除ができなければならない.

  生成・削除の要求は利用者から出される場合がある.
- プロセスが計算するのに必要な資源(メモリ,CPUなど)を各プロセスに割り当ててあげないといけない.
- 特にプロセスがCPUを使える(計算をできる)順番や時間をスケジュールしないといけない.

  一般にプロセスの数の方がCPUの数(普通1つ)より多い.

マルチプロセス

- 昨今のOSは同時に複数のプロセスを実行することができる.
  - 例えば，音楽を聞きながらワープロで文章が書ける.
- 以降では，ある時点でどんなプロセスがいくつ動作しているかを観察する.

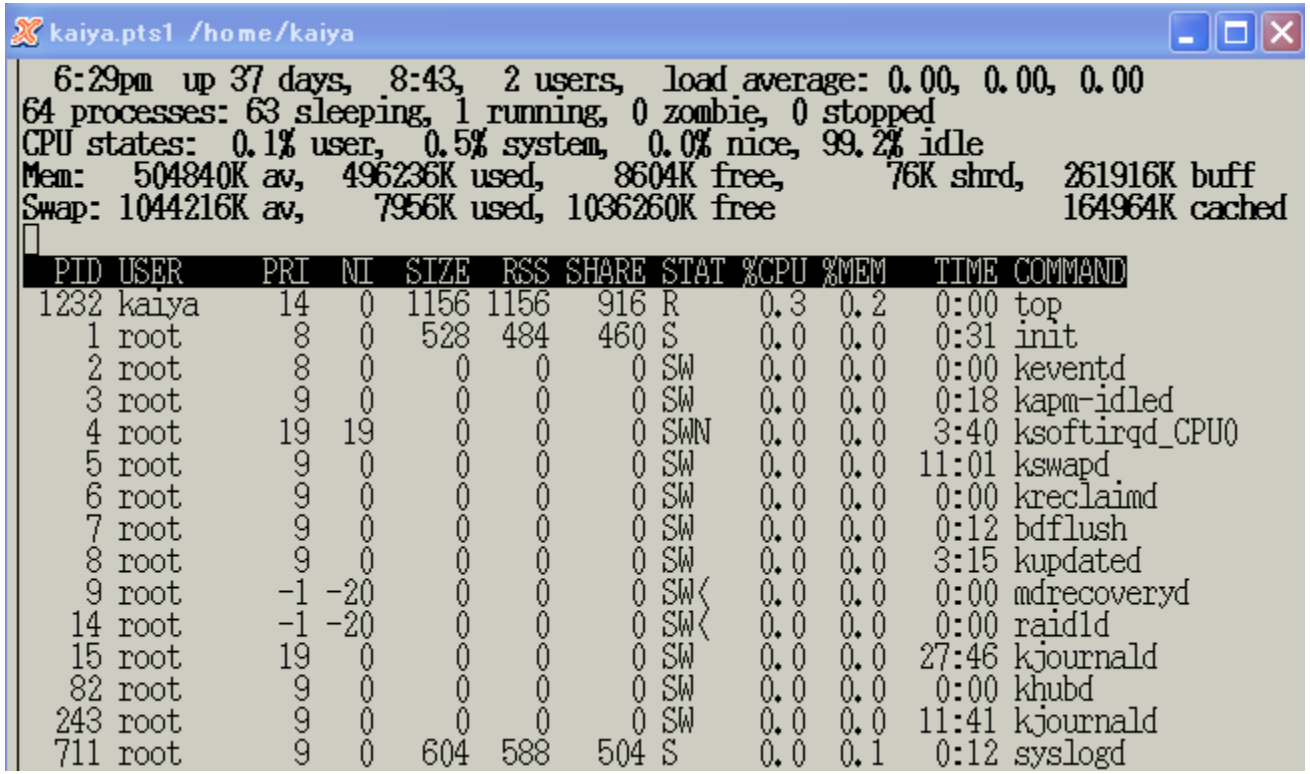
- 初回に話したTSSによって発生する現象と考えてよい.

Linuxでのプロセスの観察1

psコマンドはプロセスの状態を
安易？に観察するツールである.

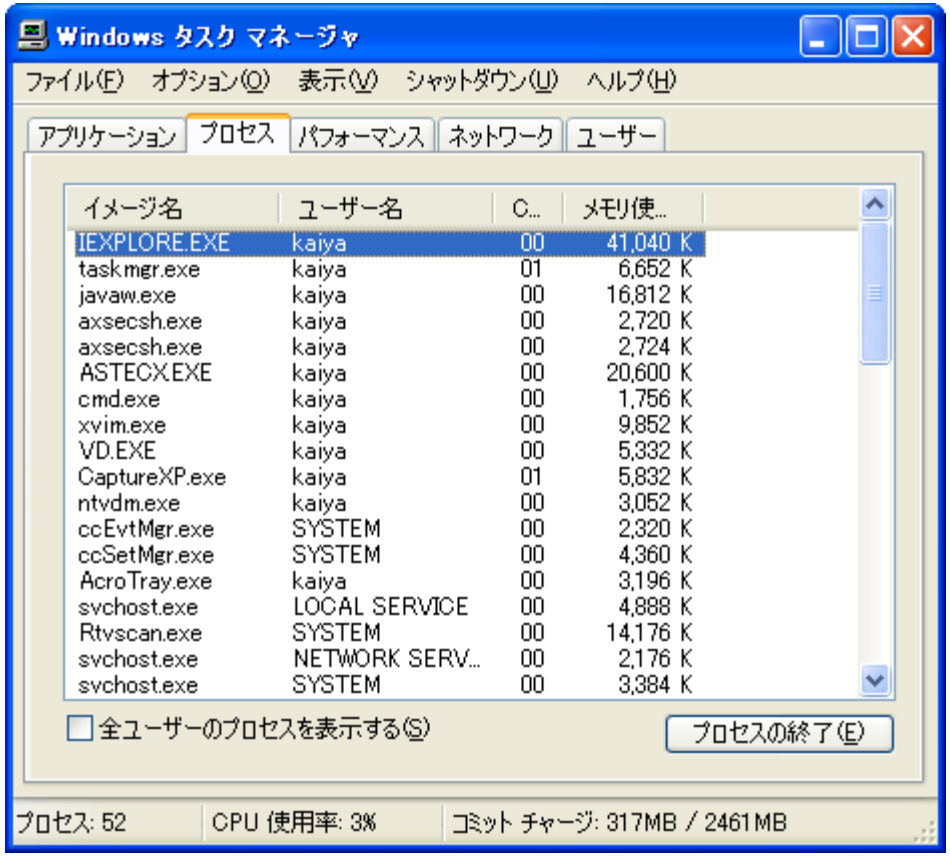
```
[kaiya@linux2001 ~]% ps x
  PID TTY      STAT   TIME COMMAND
13909 ?        S      0:07 Xvnc :1 -desktop X -auth /home/s
13921 ?        S      0:00 twm
31408 pts/0    S      0:00 -csh
31796 ttyp0    S      0:00 -bin/tcsh
31943 ttyp0    T      0:00 vi a.c
31949 ttyp0    R      0:00 ps x
[kaiya@linux2001 ~]%
```

Linuxでのプロセスの観察2



top というコマンドでも観察できる.

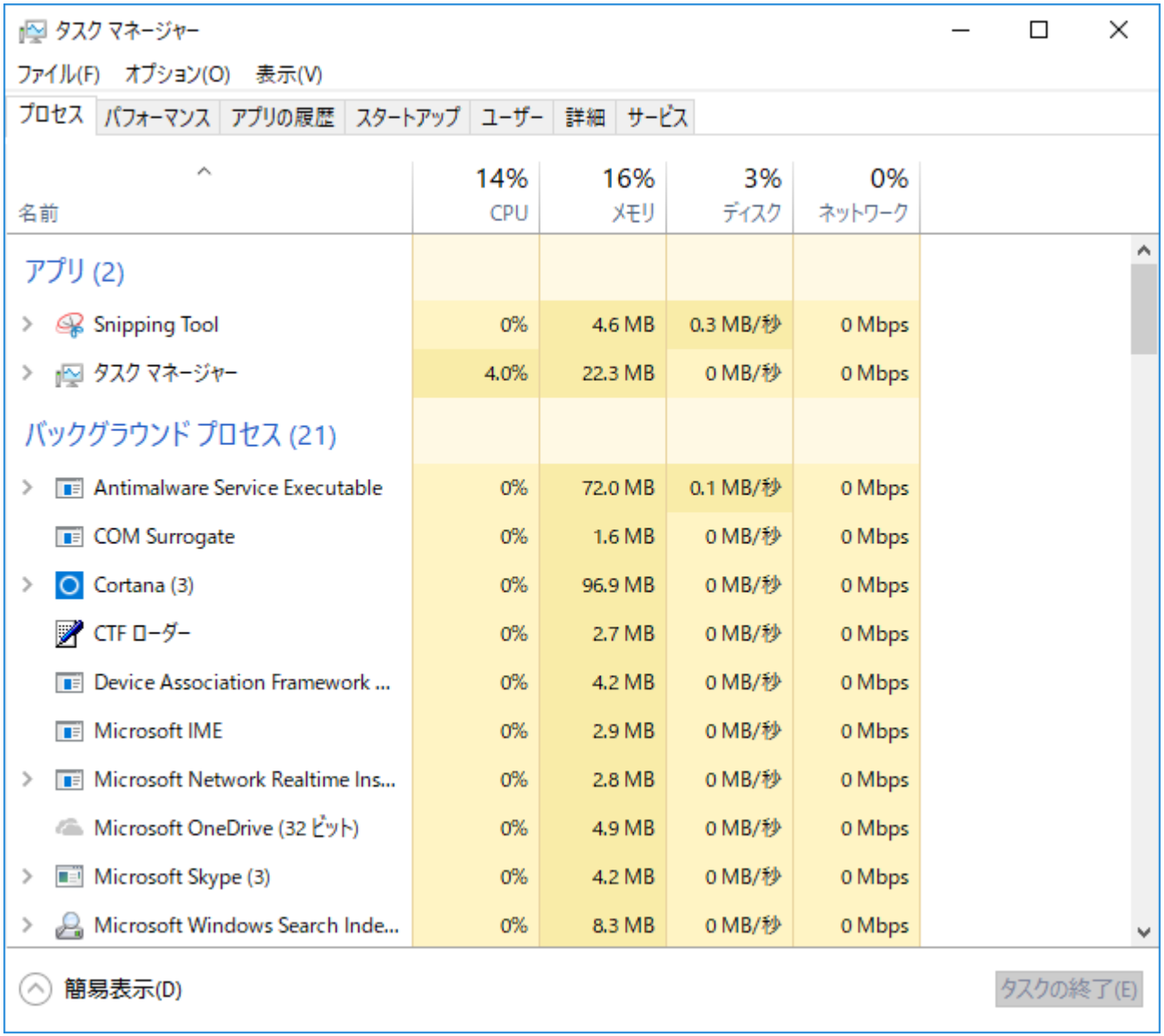
Win7でのプロセスの観察



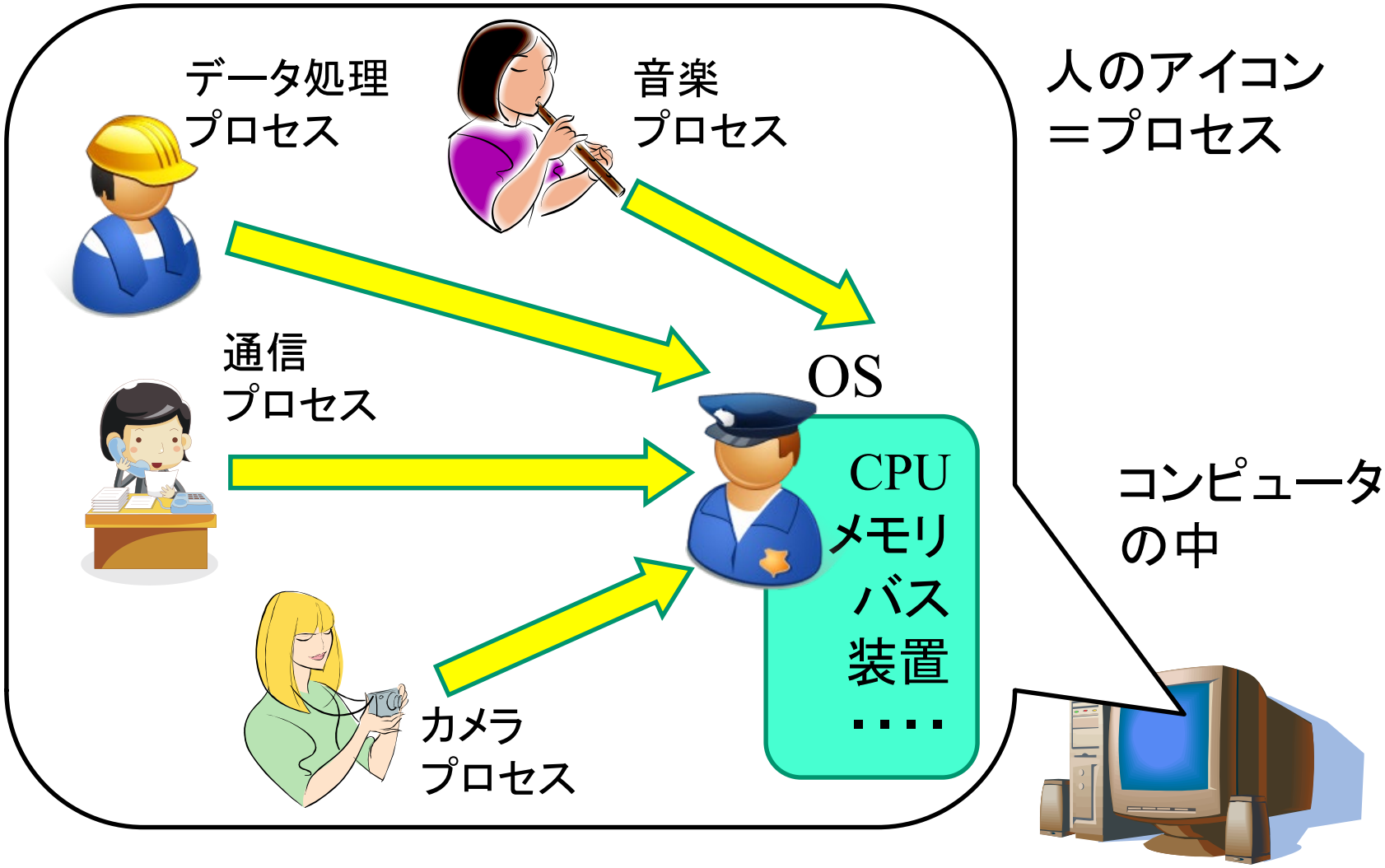
タスクマネージャーから動作しているプロセスを観察できる.

(通常, Ctrl・Alt・Delのキーを同時に押すと出てくる.)

Win10


タスク マネージャー
ファイル(F) オプション(O) 表示(V)
プロセス パフォーマンス アプリの履歴 スタートアップ ユーザー 詳細 サービス
名前 14% CPU 16% メモリ 3% ディスク 0% ネットワーク
アプリ (2)
Snipping Tool 0% 4.6 MB 0.3 MB/秒 0 Mbps
タスク マネージャー 4.0% 22.3 MB 0 MB/秒 0 Mbps
バックグラウンド プロセス (21)
Antimalware Service Executable 0% 72.0 MB 0.1 MB/秒 0 Mbps
COM Surrogate 0% 1.6 MB 0 MB/秒 0 Mbps
Cortana (3) 0% 96.9 MB 0 MB/秒 0 Mbps
CTF ローダー 0% 2.7 MB 0 MB/秒 0 Mbps
Device Association Framework ... 0% 4.2 MB 0 MB/秒 0 Mbps
Microsoft IME 0% 2.9 MB 0 MB/秒 0 Mbps
Microsoft Network Realtime Ins... 0% 2.8 MB 0 MB/秒 0 Mbps
Microsoft OneDrive (32 ビット) 0% 4.9 MB 0 MB/秒 0 Mbps
Microsoft Skype (3) 0% 4.2 MB 0 MB/秒 0 Mbps
Microsoft Windows Search Inde... 0% 8.3 MB 0 MB/秒 0 Mbps
簡易表示(D)
タスクの終了(E)

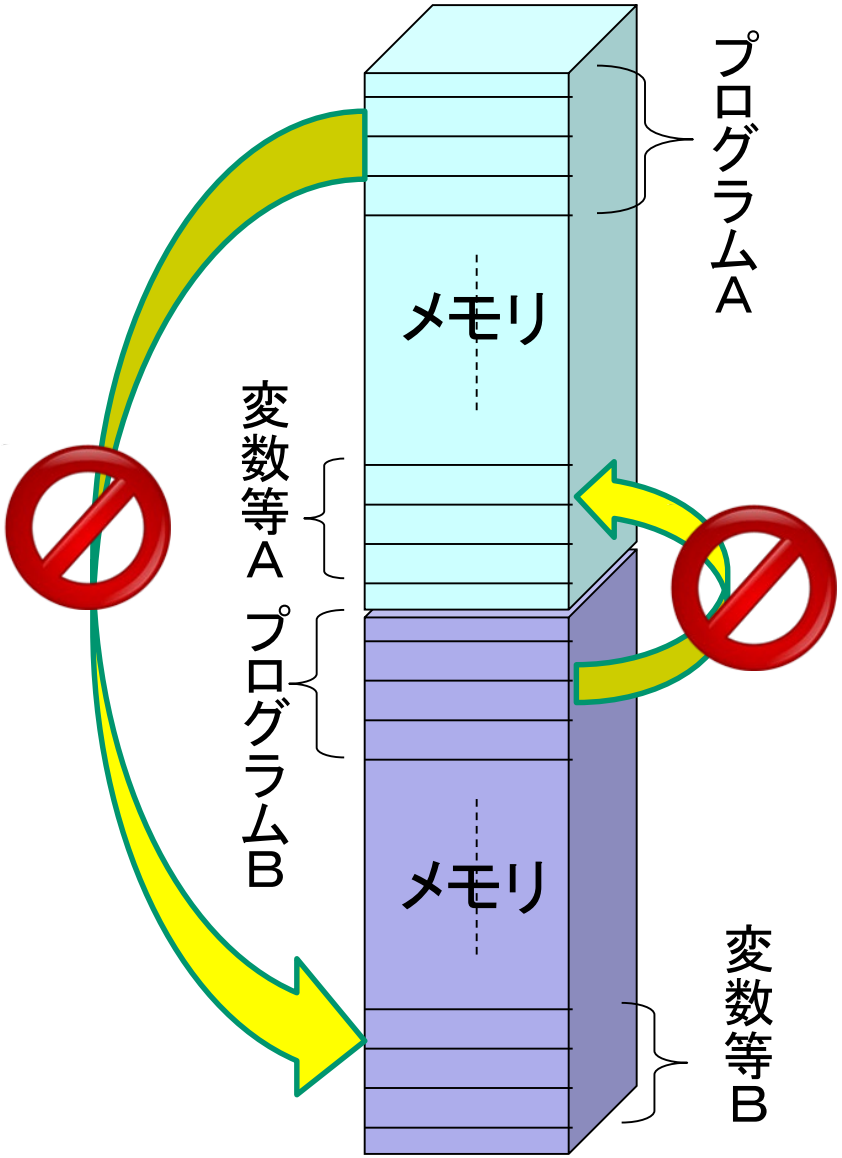
コンピュータ動作中のイメージ



OSが仲介しないとマズい例

- あるプロセスが他のプロセスのデータ等を勝手に書き換えられたら，複数プロセスが安全に同時動作できない.
  - メモリの確保はOS管理
- そもそも，勝手にメモリを占拠して，新しいプロセスを一般利用者が自由に作れたらマズい.
  - メモリのどこが利用中かはOSしか知らない.
- デバイス(例えばスピーカー等)の管理も一元的でないと不便.
  - どのデバイスをどのプロセスが使っているか覚えておかないと.
- 今さっき発売されたデバイスにアプリが自力で対応できるわけが無い.
  - OSは予めデバイスの種類を分類し，新規デバイスの影響をアプリに与えないようにしている.
- 1個(程度)しか無いCPUを長期占有されると他のプロセスが困る.
  - CPU利用に関する優先順位もOS管理.

マズい例: プロセスの相互干渉



- 個々のプロセスが自由にメモリにアクセスできたら，左のようなことが起こってしまう.
- そうならないように，メモリ利用はOSを仲介して行うことになっている.
- システムコールの必要性.

スマフォのアプリ

- スマフォの「アプリ」という用語は以下の両方を指している（場合が多い）.

1. インストールされている実行可能プログラム（アプリケーション）
2. 実行中のプログラム（プロセス）

- OSでは，上記二つは異なるモノなので，スマフォ感覚でアプリという用語は使わないほうがよい.

用語の整理

- プログラム
  - C言語等のソースコードと実行可能プログラムの双方を指すので，曖昧.
- 実行可能プログラム (ロードモジュール)
  - マシン語で書いた命令が入っているファイル，nantoka.exe 等.
  - このファイルをOSが読んで記述された命令を実行することで，プロセスを発生させる.
- アプリケーション
  - 一般には実行可能プログラムと同じ意味.
  - スマフォの場合，プロセスのことも指すような気がする.
- アプリ
  - アプリケーションのこと
- プロセス
  - 前述の通り，実行可能プログラムに従い，行われている処理.

プログラミングインタフェース

プログラミングインタフェースの目的

- 前述のようにOSが管理しなければならない資源等へのアクセスのための手段として，OSのプログラミングインタフェースがある．
- 通常，プログラミング言語のライブラリ関数として提供される．
- 狭義にはシステムコールと呼ばれる．
- アプリのプログラマは，ある関数がシステムコールか否かを意識することは少ない．

API

- Application Programming Interface
- 通常，以下に分類される.
  - システムコール
    - 前述のもの
  - それ以外
    - 基本，OSの話とあまり関係ない.
    - 単純な計算や，システムコールを組み合わせて使いやすくした関数等.
    - 特定分野の計算を行う関数群も含まれる: 画像加工，データ変換，統計処理等.

APIの使い方

- マニュアルを見て，機能を確認し，必要な引数を与えてプログラム中で呼べばよい.
- システムコールに限ったことではないが，APIを呼んでも(使っても)実行が失敗する場合がある.
  - OSの判断で実行させない場合がある.
  - 例えば，他のプロセスが管理するデータを上書きしようとする等.

```
[kaiya@flute03 ~]$ man 2 read
```

kaiya@flute03:~

```
READ(2)                    Linux Programmer's Manual                    READ(2)

名前
       read - ファイル・ディスクリプターから読み込む

書式
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t count);

説明
       read() はファイル・ディスクリプター (file descriptor) fd から最大 count
       バイトを buf で始まるバッファーへ読み込もうとする。

       count が 0 ならば、 read() は 0 を返し、他に何も起きない 。 count が
       SSIZE_MAX より大きければ、結果は特定できない。

返り値
       成功した場合、読み込んだバイト数を返す (0 はファイルの終りを意味する)。
       ファイル位置はこの数だけ進められる。この数が要求した数より小さかった とし
        てもエラーではない; 例えば今すぐには実際にそれだけの数しかない場合 (
       ファイルの最後に近いのかもしれないし、パイプ (pipe) や端 末 (terminal)
       か ら読み込んでいるかもしれない) や read() がシグナル (signal) によって
       割り込まれた場合にこれは起こりえる。エラーの場合は、-1 が返され、 errno
       が 適切に設定される。この場合はファイル位置が変更されるかどうかは不定で
       ある。

エラー
       EAGAIN ファイル・ディスクリプター fd がソケット以外のファイルを参照して
              いて、非停止 (non-blocking) モード (O_NONBLOCK) に設定されており
              、読み込みを行うと停止する状況にある。

       EAGAIN または EWOULDBLOCK
              ファイル・ディスクリプター fd がソケットを参照していて、非 停 止
:
```

マニュアルの内容

- 書式
  - インクルードすべきヘッダー
  - 関数名
  - 返り値の型
  - 引数の数と型
- 説明
  - 関数の意味が文章で説明してある.
- 返り値
  - 返り値の意味が文章で説明してある.
- エラー
  - エラー値の種類と意味が説明されている.

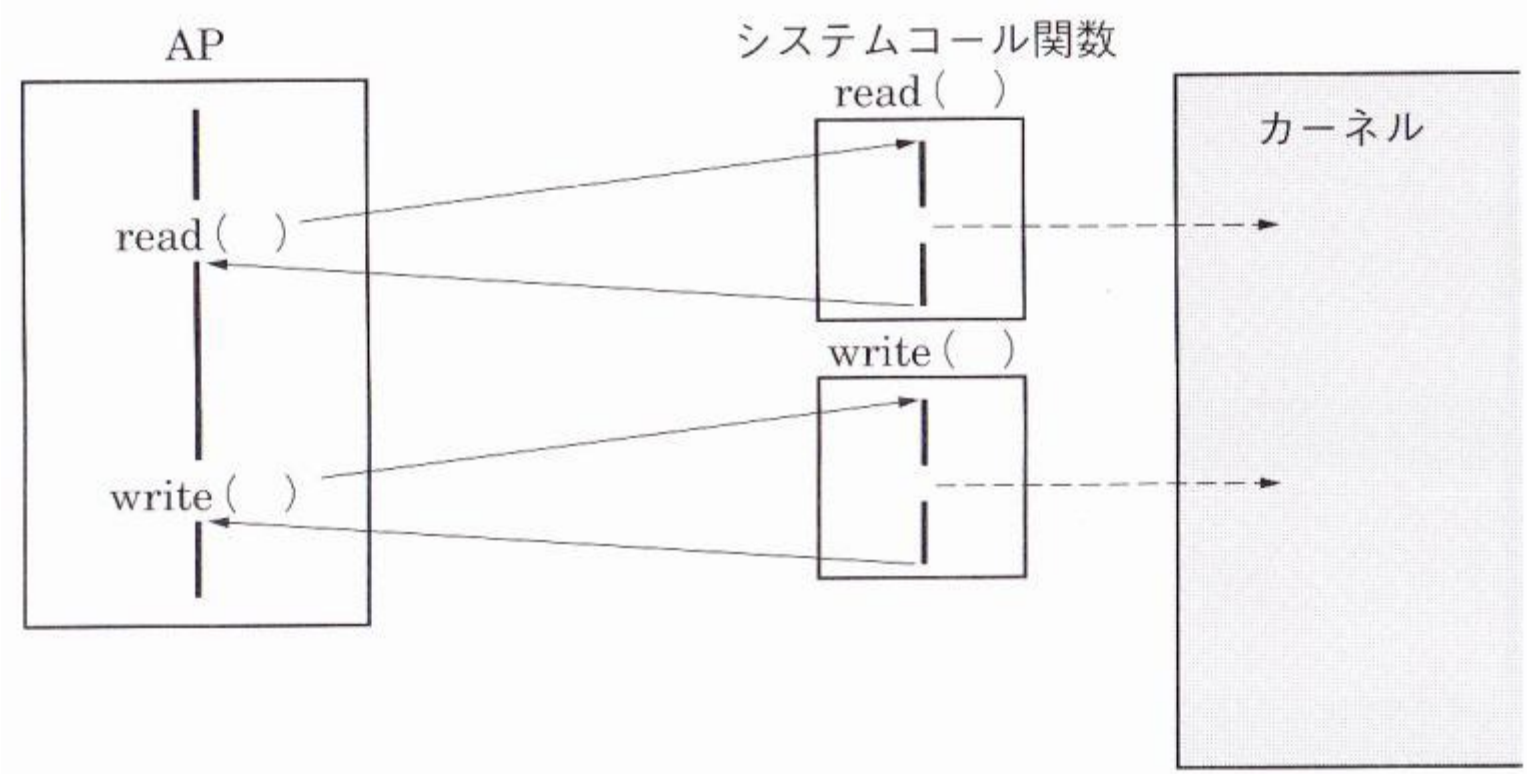
システムコールの直接・間接利用

両方，キーボードから1行文字を読んで，表示するプログラム．

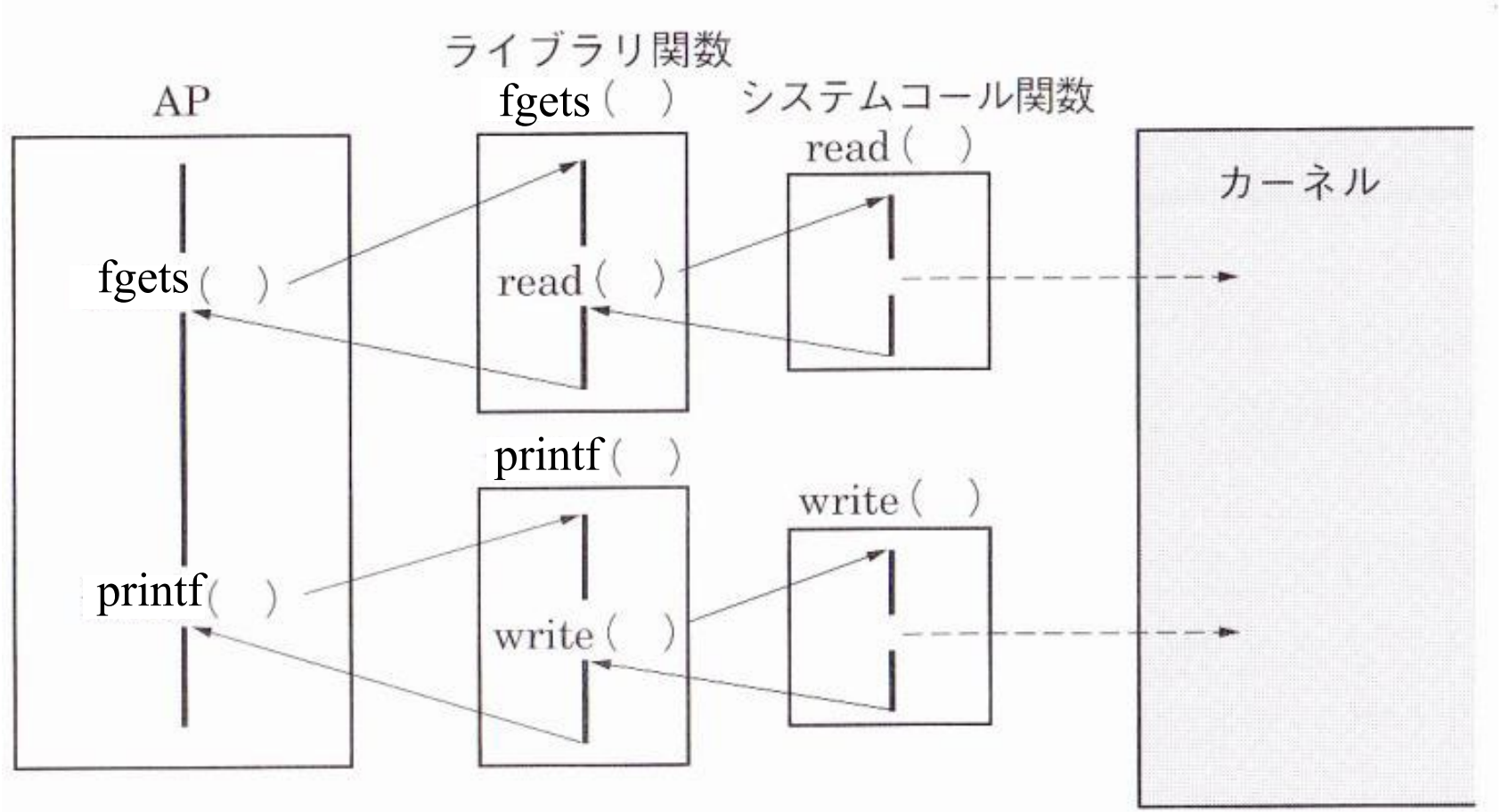
```
// システムコール利用
 int main(void){
char buf[1];
    while(0 < read(0, buf, 1)){
        write(1, buf, 1);
        if(buf[0]=='¥n') break;
    }
}
```

```
// 標準関数利用
#include <stdio.h>

int main(void){
char buf[100];
    if(fgets(buf, 100, stdin)!=NULL)
        printf("%s", buf);
}
```

システムコールを直接使うと，ループと改行認識が必要．
このシステムコールには改行を特別扱いする機能は無い．



（a）応用プログラム（AP）からシステムコール関数を利用

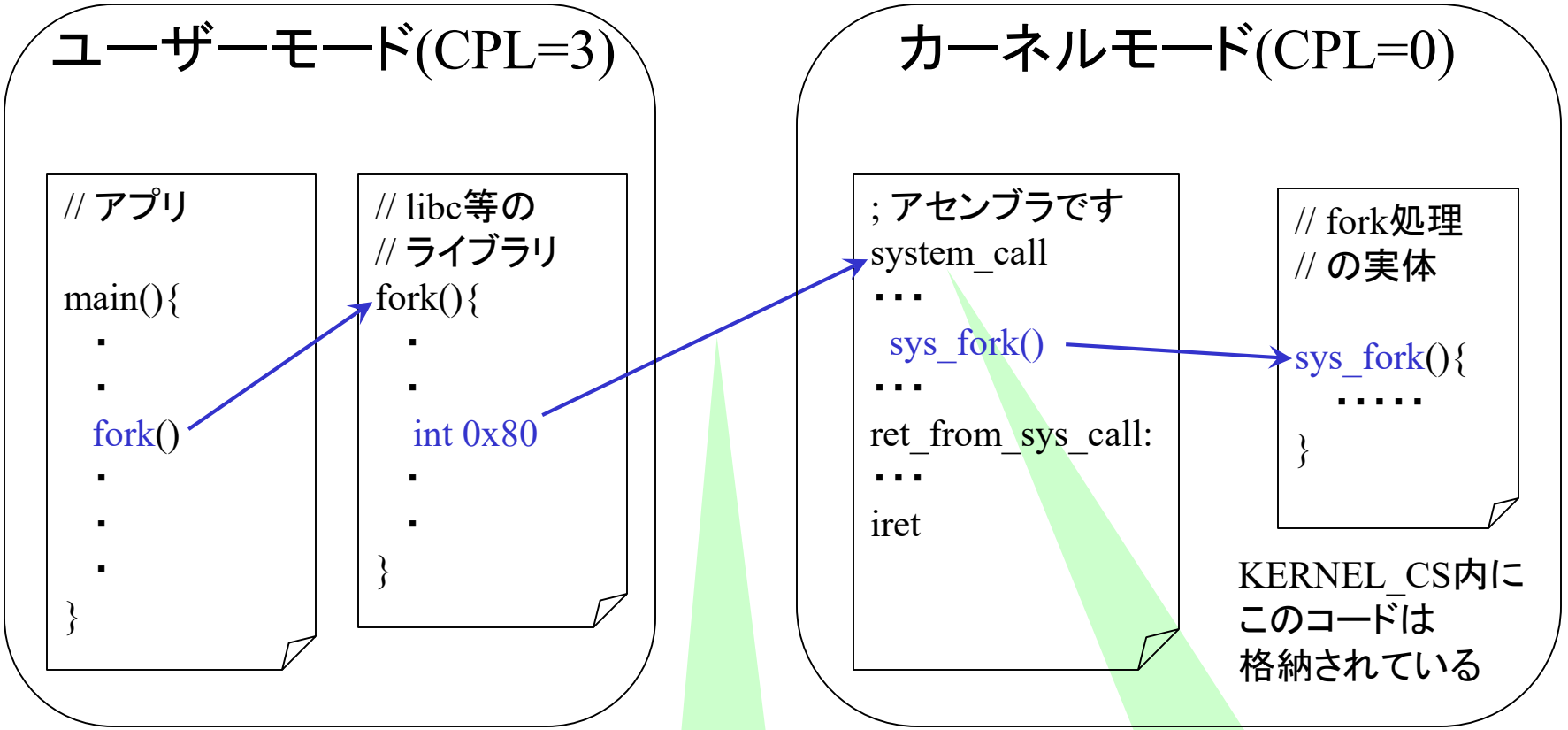


（b）応用プログラム（AP）からライブラリ関数を利用

システムコールによる権限昇格

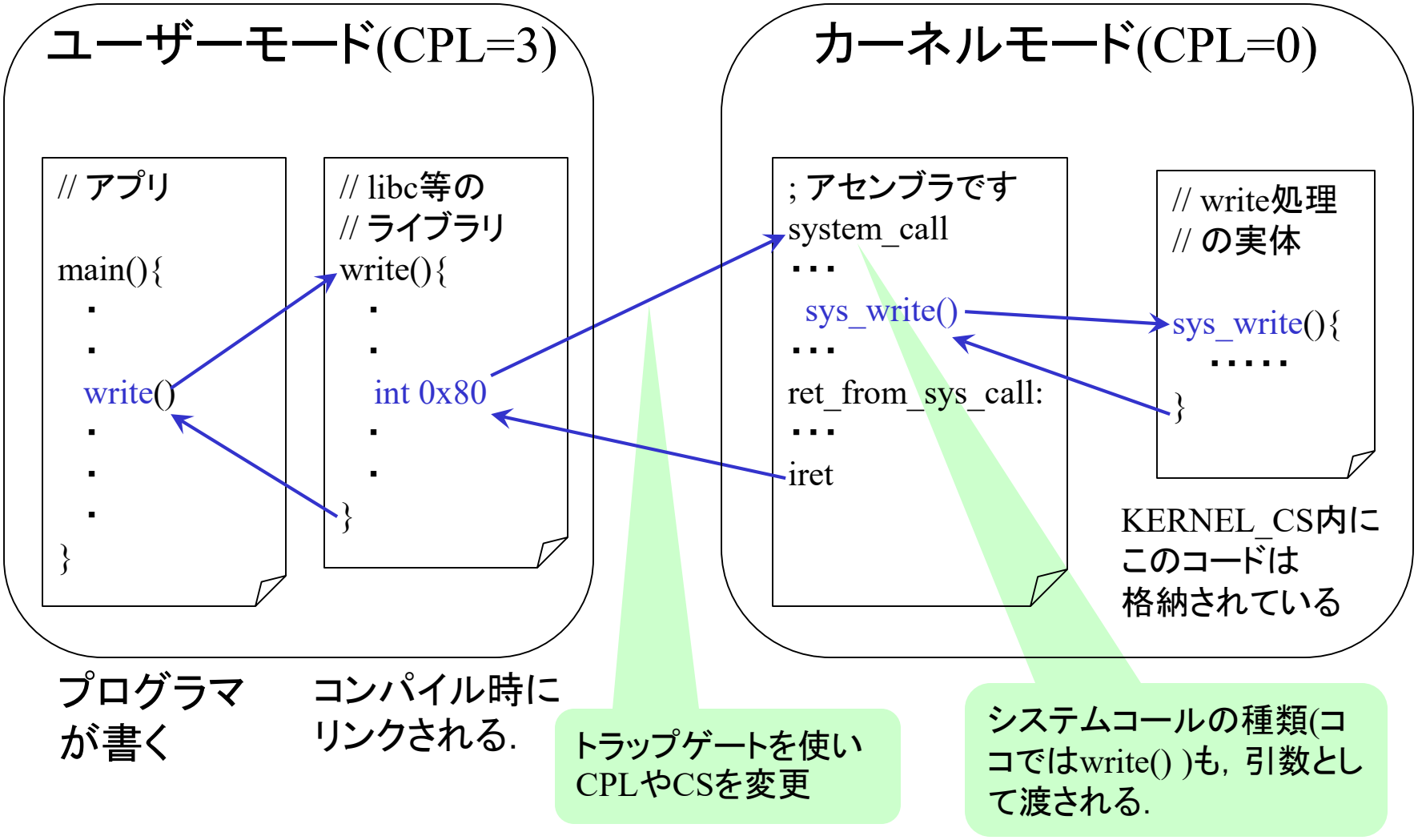
- アプリがシステムコールを呼んだ場合でも，OSが独占的に管理している資源を使うことができる.
- 要はシステムコールが実行されている間は，アプリも高い権限で動作している.
- システムコール終了後に元の低い権限にもどすための仕組がOSにはある.
  - 多くのCPUでは割り込みという機構を用いる.
  - 詳細は次回.

例えばfork()



※ fork()は返らないので不適切な例でした.

例えばwrite()


ユーザーモード(CPL=3)
カーネルモード(CPL=0)
// アプリ
main(){
write()
}
// libc等の
// ライブラリ
write(){
int 0x80
}
; アセンブラです
system_call
sys_write()
ret_from_sys_call:
iret
// write処理
// の実体
sys_write(){
}
KERNEL_CS内に
このコードは
格納されている
プログラマ
が書く
コンパイル時に
リンクされる.
トラップゲートを使い
CPLやCSを変更
システムコールの種類(コ
コではwrite())も，引数とし
て渡される.

カーネル(Kernel)とは

- OS の核になる部品のこと．一般には以下を管理する部分．
  - プロセス
  - メモリ
  - デバイス
  - ファイルシステム・・・コレはちょっとグレイ
- 一方，以下はカーネルに入らない．
  - ユーザーインタフェース
    - Shell とか デスクトップとかはカーネルではない
  - サーバー・サービス群
    - ウエブサーバー メールサーバー等
  - ユーザー管理・・・ ちょっとグレイ

具体的なAPI: システムコール

- ファイル関係 デバイス関係
  - open(), close(), read(), write(), lseek()
  - ioctl()
- プロセス関係
  - fork(), kill(), exec(), exit(), wait()
- メモリ関係
  - brk()

システムコール以外の関数

- ファイル関係
  - fopen, fclose, remove, rename
- プロセス関係
  - popen, pclose
- メモリ関係
  - malloc, calloc, free

Linuxの場合

- マニュアル 2章にシステムコールがまとめられている.
  - man 2 intro
  - man 2 syscalls
- それ以外の標準APIは3章にまとめられている.
  - man 3 intro
- 第三者(オープンソース開発者等)が提供するAPIについては特に規定は無い.

man 2 intro

kaiya@flute03:~

```
INTRO(2)                    Linux Programmer's Manual                   INTRO(2)

名前
       intro - システムコールの説明

説明
       マ ニュアルの 2 章では Linux のシステムコールについて説明している。シス
       テムコールは Linux カーネルへのエントリポイントである。通常は、システム
       コ ールは直接起動される訳ではなく、ほとんどのシステムコールには対応する
       C ライブラリのラッパー関数があり、そのラッパー関数がシステムコールを 呼
       び 出すのに必要な処理を実行する。そのため、システムコールを呼び出すのは
       通常のライブラリ関数を呼び出すのと同じように見える。

       Linux のシステムコールのリストについては syscalls(2) を参照のこと。

返り値
       エラーの場合、ほとんどのシステムコールは負のエラー番号を返す (具体的 に
       は  errno(3) で説明されている定数のいずれかを負にした値を返す)。 C ライ
       ブラリのラッパーは呼び出し元からこうした詳細を隠蔽している。システム コ
       ールが負の値を返した場合、ラッパーは絶対値を errno 変数にコピーし、ラッ
       パーの返り値として -1 を返す。

       成功時にシステムコールが返す値はシステムコールにより異なる。多くのシ ス
       テ ムコールは成功時に 0 を返すが、成功時に 0 以外の値を返すシステムコー
       ルもある。詳細は個々のマニュアルページで説明されている。

       マニュアルページの「書式」の節に規定されたヘッダファイルから関数の定 義
       を 得るために、プログラマが機能検査マクロを定義しなければならない場合が
       ある。このような場合には、必要なマクロがマニュアルページで説明されて い
       る 。機能検査マクロのさらなる情報については、 feature_test_macros(7) を
       参照のこと。

準拠
       この章の関数が準拠する Unix システムや標準を示すのにいくつかの単語や 略
:
```

man 2 syscalls

kaiya@flute03:~

```
SYSCALLS(2)                Linux Programmer's Manual                SYSCALLS(2)

名前
       syscalls - Linux のシステムコール

書式
       Linux のシステムコール。

説明
       システムコールは、アプリケーションと Linux カーネルとの間の基本的なイン
       タフェースである。

   システムコールとライブラリのラッパー関数
       システムコールは一般には直接起動されず、 glibc (や他の何らかのライブ ラ
       リ)   経 由 で 起 動される。システムコールの直接起動については、詳細は
       intro(2) を参照のこと。いつもという訳ではないが、普通は、ラッパー関数の
       名前はその関数が起動するシステムコールの名前と同じである。例えば、glibc
       には truncate() という関数があり、この関数は "truncate" システムコー ル
       を起動する。

       た いていの場合、glibc のラッパー関数はかなり簡単なもので、システムコー
       ルを起動する前に引き数を適切なレジスタにコピーし、システムコールが返 っ
       た後は errno を適切に設定する以外は、ほとんど処理を行わない (これらは、
       ラッパー関数が提供されていない場合にシステムコールを起動するのに使用 す
       る syscall(2) により実行される処理と同じである)。 [注意] システムコール
       は失敗を示すのに負のエラー番号を呼び出し元に返す。失敗が起こった際に は
       、 ラ ッパー関数は返されたエラー番号を反転して (正の値に変換し)、それを
       errno にコピーし、ラッパー関数の呼び出し元に -1 を返す。

       しかしながら、時には、ラッパー関数がシステムコールを起動する前に何ら か
       の 追加の処理を行う場合がある。例えば、現在、二つの関連するシステムコー
       ル truncate(2) と truncate64(2) があり、glibc のラッパー関数 truncate()
       は 、カーネルがこれらのシステムコールのうちどちらを提供しているかをチェ
       ックし、どちらを採用するかを決定する。
:
```

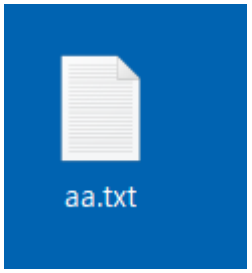
man 3 intro

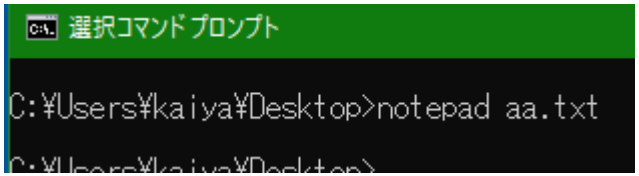
kaiya@flute03:~

```
INTRO(3)                   Linux Programmer's Manual                  INTRO(3)

名前
       intro - ライブラリ関数の紹介

説明
       マ ニュアルの 3 章では、システムコールを実装した 2 章で説明されたライブ
       ラリ関数（システムコールのラッパー）を除いた全てのライブラリ関数につ いて説明している。

       この章で説明している関数の多くは標準 C ライブラリ（libc）のものである。
       また、いくつかの関数は、他のライブラリ（例えば、数学ライブラリ の libm
       やリアルタイムライブラリ librt ）のものである。後者の場合は、マニュアル
       ページに、必要なライブラリとリンクするために必要なリンカオプションが 示
       さ れ ている（例えば、前述のライブラリの場合はそれぞれ -lm や -lrt であ
       る）。

       マニュアルページの「書式」の節に規定されたヘッダファイルから関数の定 義
       を 得るために、プログラマが機能検査マクロを定義しなければならない場合が
       ある。このような場合には、必要なマクロがマニュアルページで説明されて い
       る 。機能検査マクロのさらなる情報については、 feature_test_macros(7) を
       参照のこと。

準拠
       この章の関数が準拠する Unix システムや標準を示すのにいくつかの単語や 略
       号が使用されている。 standards(7) を参照のこと。

備考
   著者と著作権
       著 者と著作権の状態はマニュアルページのヘッダを見ること。これらはページ
       毎に異なる可能性があることに注意。

関連項目
       intro(2), errno(3), feature_test_macros(7), libc(7), standards(7)
:
```

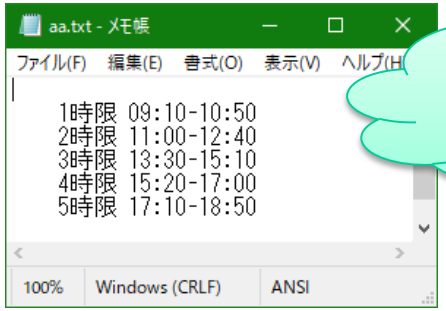
プロセスの呼び出し

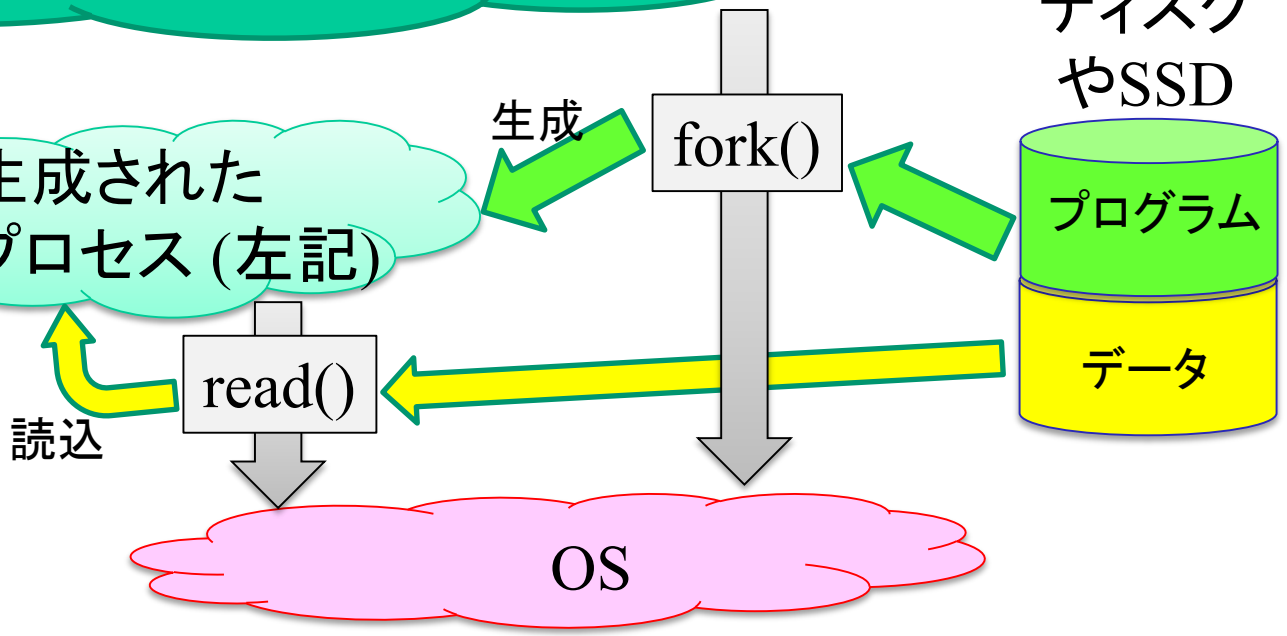
- ユーザー入力の受付法
  - アプリやファイルのアイコンをクリック
  - コマンドを打つ
- ユーザー入力の処理
  - プログラムが格納されたファイルを特定
  - 必要ならデータファイルも特定
- プログラムからプロセスの生成
  - プログラムをメモリにロード，fork()の利用
  - 必要ならデータを読む read()の利用





プロセスを新規に生成することのできる
プログラムのプロセス (上記)





細かな用語説明

- ラッパー
  - Wrapper「包むもの」という意味
  - システムコールを包んで，より使いやすくしたものという意味合いで使われている.
  - アプリで利用する read 等の関数はそもそもラッパーであり，システムコールの実体はOSのプログラムの一部として記述されている.
- ストリーム
  - Stream「流れ」という意味.
  - 入出力装置等とやり取りするデータを，一連の連続列として扱う考え方.
  - 一連の列なので，順々に取り出す(送り出す)処理を適用できる.
- バッファー
  - Buffer「緩衝(器)」という意味.
  - 細切れのデータを途中で一定量蓄える場所(メモリ領域)のこと.
  - fputs, fgets 等の関数が行単位で処理が行えるのはコレのおかげ.

互換性

- 互換性 compatibility は曖昧であるため現在はあまり使わない.
- Interoperability と replaceability が代りに使われる.
- Interoperability 相互運用性
  - 指定されたシステムとちゃんと相互作用できるかどうかの特性.
  - 例: ウエブサーバーは相手がIEでもFFでもちゃんと動く等.
- Replaceability 置換性
  - 置き換えても平気かどうかの特性.
  - 例: ウエブサーバーapache を nginx に置き換えてもIEでちゃんと動く.
- これらの用語は ISO9126 で規定されている.

APIでの相互運用性と置換性

- 相互運用性はAPIを使う側が変わっても，ちゃんと動くという概念.
  - 例えば，LinuxでもWinでも，同じ関数を使って，同じような動作をするか等.
- 置換性はAPIの中身(実装)がかわっても，ちゃんと動くという概念.
  - 例えばメーカー製のAPIを使っても，オープンソースのAPIに置き換えてもちゃんと動くという意味.

アプリの相互運用性

- 通常，実行可能プログラム(.exe 等)では，相互運用性は低い(というか無い).
  - WinのexeはLinuxでは直接は動かない.
- ソースコード (.c .cxx 等)の場合，相互運用性がある程度高い場合が多い.
  - とはいえ，ところどころ，場合分け等が必要.
- インタプリタ型言語(Java, Ruby, Python等)のプログラムは，相互運用性は高い.
  - インタプリタ側がOSの違いを吸収しているため.

移植性

- Portability
  - ソフトウェア製品を変換して，ある環境から他の環境に移す際の性能.
  - 例えば，MS Office を Windows から Mac に移す等.
  - 何もしないで移植できればソレにこしたことは無いが，大抵，何か直しが必要.
- こちらも ISO9126という国際標準で定められている用語.

本日は以上

アンケートのほう，
よろしくご提出ください